MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.036—Introduction to Machine Learning
Spring 2015

Project 2: Clustering Works of Art   Issued: Fri., 3/13 Due: Fri., 4/03 at 9am

Project Submission: Please submit a .zip file of your project on the Stellar web site by 9am, 4/03. The zip file should contain: a PDF of your project report and your python code.

Introduction

For this project we will use a small part of the huge Google Art Project[1] that we curated to contain distinctive works of 10 different artists (*Canaletto, Monet, Romney, Turner, Cozens, Cézanne, Gaugin, Rubens, Rembrandt, Wilson*). We provide you with a different number of artworks (images) per artist, ranging between 20–70, for a total of 415 images. Because the images were originally of different sizes, we have resized them to fit into 128×128 pixels (padded with black pixels to maintain original aspect ratios). This setting is modeled after a real-world use of machine learning: analysis of complex images with large variability in content and style. The goal of this project is to see through this complexity and uncover some underlying patterns that can be used to solve prediction tasks. You will use different image features to cluster artworks in an unsupervised way.

Images are high-dimensional objects, typically represented as thousands (in this project) to millions (more generally) of pixel values. You will explore how to reduce high-dimensional vectors of pixels to a low-dimensional vector of features. You will also think about data representation, by experimenting with different popular image features.

You will implement dimensionality reduction through PCA. You will also implement two clustering techniques covered in class: K-means and K-medoids. You will experiment with clustering in both unsupervised and semi-supervised settings: (i) to explore what clusters your algorithms can discover based on different image features used; and (ii) to test how to use clustering to solve a classification problem in the face of very little labeled data?

Other than your code, you will submit a PDF with relevant plots and discussion. What to put in the PDF for each part is indicated with **“include”** annotations below.

[1] `<https://www.google.com/culturalinstitute/artists?projectId=art-project>`

0. Setup

As with the last project, please use Python's NumPy numerical library for handling arrays and array operations; use `matplotlib` for producing figures and plots.

Download `project2.zip` from Stellar and unzip it into a working directory. The zip file contains image features and toy data files in `.csv` format, original images in `.png` format, and several relevant Python files.

Toy data

To help you get started (for visualization and debugging) you will use the toy dataset of 2D points provided in the files `toy_cluster_data.csv` and `toy_pca_data.csv`. We have provided a function `loadCSV` for reading these files into your program. You will do most of the coding for this project in this section. The code you develop here will be used for the other parts of the project.

1. **Dimensionality Reduction via PCA**

   (a) Write a Python function `ml_split` that centers the data (via mean subtraction). Your input should be:

   - `X`, an $n \times d$ Numpy array of n data points, each with d features.

   Your function should return a tuple `(X_centered, mean)`, where:

   - `X_centered` is an $n \times d$ Numpy array, such that $\texttt{X\_centered}[i, j] = \texttt{X}[i, j] - \texttt{mean}[j]$.
   - `mean` is a $d \times 1$ Numpy array, such that:

   $$\texttt{mean}[j] = \frac{1}{n}\left[\sum_{0 \le i \le n-1} \texttt{X}[i, j]\right], \ \forall j : 0 \le j \le d - 1.$$

   (b) Write a Python function `ml_compute_eigenvectors` that computes the principal component directions of a (centered) data matrix X. This function should first compute the covariance matrix, $X^T X$, and then find its eigenvectors. Your inputs are:

   - `X`, an $n \times d$ Numpy array of n data points, each with d features.
   - `m`, the number of principal components to compute

   The function returns:

   - `U`, a $m \times d$ matrix whose rows are the first m PCA directions i.e. top m eigenvectors of $X^T X$, in descending order of eigenvalues

   Hint: you can use Numpy's `eig` function to compute eigenvalues and eigenvectors (but note that the eigenvectors still need to be sorted!)

   (c) Write a Python function `ml_pca` that implements PCA dimensionality reduction of a (centered) data matrix X. Your input should be:

   - `X`, an $n \times d$ Numpy array of n data points, each with d features.
   - `U`, a $m \times d$ matrix whose rows are the top m eigenvectors of $X^T X$, in descending order of eigenvalues

Your function should return a matrix `E`, where:

- `E` is an $n \times m$ matrix, whose rows represent low-dimensional feature vectors

(d) Write a Python function `ml_reconstruct`, which uses the PCA features and eigenvectors to reconstruct the original data using lower-dimensional feature vectors. Your input should be:

- `E`, an $n \times m$ Numpy array of n data points, each with m features.
- `U` is a $m \times d$ matrix whose rows are the top m eigenvectors of $X^T X$.
- `mean` is a vector which is the mean of the data.

Your function should return a matrix `X_recon`, where:

- `X_recon` is a $n \times d$ Numpy array of n reconstructed data points

(e) In this part, we will combine the functions you wrote above to find the best projection of a set of $2D$ points on a line. Your tasks are:

i) Read in the data provided in `toy_pca_data.csv`
ii) Use the function `ml_split` to generate the means vector, and center the data.
iii) Use the function `ml_compute_eigenvectors` to generate the top $m = 1$ eigenvectors.
iv) Use the function `ml_pca` to generate low-dimensional feature vectors.
v) Use the function `ml_reconstruct` to reconstruct the original points using the low-dimensional feature vectors.
vi) Use the provided function `plot_pca` to plot the original points together with the reconstructed points on a single scatter plot.

To see if your results make sense, we provide you with an example of a PCA plot on a different set of data. This example can be found in `pca_example.png`: the red points are the original 2D data, and the green points are the 1-dimensional reconstructions.
Include: The plot of the reconstructed points with $m = 1$. Also give your interpretation of the reconstructed points.

2. **Clustering**

(a) Write a Python function `ml_k_means` that implements K-means clustering. Your input should be:

- `X`, an $n \times d$ Numpy array of n data points, each with d features.
- `k`, the number of desired clusters
- `init`, a $k \times d$ Numpy array of k data points, each with d features, which are the initial guesses for the centroids.

Your function should return a tuple `(centroids, clusterAssignments)`, where:

- `centroids` is a $k \times d$ Numpy array of k points, each with d features, indicating the final centroids.
- `clusterAssignments` is an $n \times 1$ Numpy array of integers, each between 0 and $k - 1$, indicating which cluster the input points are assigned to.

Run your `ml_k_means` function on the data provided in `toy_cluster_data.csv`. Use the first k points in the file as your initialization for the centroids. This initialization is used for simplicity with this toy dataset. Plot the final clusters you obtain with $k = [2, 3, 4]$ using the function `plot_2D_clusters`, which is provided to you.

Include: 3 plots of clustering for the different values of k. Also comment on why the initialization we suggested might be a poor choice in general, and a random initialization might more preferable.

(b) Write a Python function `ml_k_medoids` that implements K-medoids clustering. Your input and output variables should be the same as for your `ml_k_means` function, but we shall refer to the final cluster centers as 'medoids' instead of 'centroids'. Note that the centroids you generate with K-means might not correspond to any of the original data points in X, but the medoids generated by K-medoids do correspond to actual data points. Run your `ml_k_medoids` function on the data provided in `toy_cluster_data.csv`, using the first k data points as your initial medoids. Plot the final clusters you obtain with $k = [2, 3, 4]$, using the same plot function as above.
Include: 3 plots of clustering for the different values of k. Also comment on why having medoids correspond to actual data points might be useful.

PCA and image reconstruction

In this question, you will visually explore the images and see the effects of PCA on the image data.

3. First, to get familiar with the image data, use the `plotArtworks` function provided to display the artworks, grouped by artist. Note that we can plot images grouped by artists because we have the artist labels for all the artworks, which may not necessarily be the case in general. In other words, what you see could be considered as the ground-truth clusters, and we would hope that our algorithm discovers these clusters (without any knowledge of the labels). Take some time to examine the artworks: is there a clear separation between artists? Would you be able to figure out which artworks belong to which artist if you didn't know the labels? Where might you make confusions? Which artists are more similar to each other? Which artists stand out from all the rest?
Include: a few sentences reporting your qualitative assessment of the data.

4. Use the `loadPixelFeatures` function provided to you to read in the image features for all the artworks into a data matrix `X`. These features correspond to taking the raw grayscale pixel values of an image[2] and concatenating them into one long vector. Now you will run PCA on this data matrix to produce a matrix `E` of principal eigenvectors for the artworks. Please use the provided function `ml_compute_eigenvectors_SVD` to compute the eigenvectors of $X^T X$ instead of the function `ml_compute_eigenvectors` that you wrote in the first part of the assignment. Computing the SVD (singular value decomposition) of `X` is more feasible than computing the full covariance matrix, especially with high-dimensional data, and both give you the eigenvectors of $X^T X$. In practice, SVD is the preferred method for eigendecomposition of $X^T X$, rather than computing eigenvectors. You can read more about this in the appendix.

[2]We use grayscale pixel values for simplicity, and to be able to visualize the PCA results in a meaningful way without worrying about color spaces and representations.

5. Now, explore the effect of using more or fewer dimensions to represent images.
   Perform PCA, project the original data into the lower-dimensional space, and then reconstruct high-dimensional images from the lower dimensional projections. You can plot a gallery of the reconstructions of selected images using the provided `plotGallery` function. For plotting, we provide you a selection of 5 images per artist for all 10 artists in `labels.csv`. Try varying the parameter m and examining the reconstructions for this selection of images. Which artists are already discriminative (different from the rest) at $m = 10$? Include the reconstructions for the selected images for $m = 10$ in your write-up. What value of m is required to start differentiating the painted landscapes from each other (when do they stop all looking the same)? Include the reconstructions for this value of m.
   Include: Reconstructions for a few values of m and a short discussion to answer the above questions.

Clustering images

Here you will apply your clustering algorithms to the image data. One question that immediately arises when using K-means or K-medoids is how to choose k. A common way to choose k is to evaluate the quality of the clustering for various k and pick the k that results in the best score. Cluster quality can be evaluated using two different methods: internal and external evaluation. In internal evaluation the clustering result is evaluated based on features of the data that were used for obtaining the clustering. In external evaluation, clustering results are evaluated based on (label) information that was not available to the clustering algorithm.

We will first perform internal evaluation. We use the *Dunn index* which is one possible clustering evaluation metric that looks for dense clusters with maximal separation from one another. It is defined as:

$$D = \min_{1 \le i \le n} \{ \min_{1 \le j \le n, i \ne j} \{ \frac{d(i,j)}{\max_{1 \le k \le n} d'(k)} \} \},$$

where $d(i,j)$ is the inter-cluster distance between clusters i and j, and $d'(k)$ is the intra-cluster distance of cluster k. Inter-cluster and intra-cluster distances can be defined in a variety of ways, but for this project, inter-cluster distance is defined as the Euclidean distance between the centroids of the two clusters being compared, and intra-cluster distance as the maximum Euclidean distance between any pair of points in the cluster. Thus, based on how the Dunn index is defined, larger values are likely to correspond to to higher quality clusterings.

For external evaluation, we use the notion of *cluster purity*. This checks how well the clusters obtained by our algorithms correspond to the actual clustering of images by artist (using our ground-truth artist labels). We assign a label (artist) to each cluster, based on the most frequently occurring label in that cluster. This is called majority voting. Thus every artwork in that cluster gets the majority label. After this, we compute the total fraction of artworks that have been correctly labeled. Cluster purity is thus an index between 0 and 1, where higher purity indicates better clustering. Note that in real settings, the labels are most likely unknown and external evaluation is not possible. In these cases, we would rely on internal evaluation measures and qualitative evaluations to help us select a good clustering.

6. For each value of $m = [10, 50, 400]$ (you can also try other values of m like 20, 100, etc.):
   - perform PCA on the data matrix X
   - reconstruct the data matrix using only m dimensions to produce a new data matrix E
   - run your K-means function on E with an initialization provided by `init_medoids_plus` and different values of $k = [2, 5, 10]$ (you can also try other values of k)

- plot the clusters you obtain for each value of k using the `plotClusters` function
- compute the Dunn index using the provided function `computeDunnIndex` for k
- compute the cluster purity using the provided function `computeClusterPurity` for each k

Thus, if you work with 3 values of m and 3 values for k, you should produce 9 different clusterings. For each clustering, qualitatively observe your clustering results, report the Dunn index and cluster purity value. How do the clusters, and cluster purity vary with k? How do the clusters, Dunn index, and cluster purity vary with m? If m is very small (e.g. 10), what happens when k is small/large? (think back to your experiments with PCA from part 2). Note that the Dunn Index is primarily designed to choose between algorithms, which in this case would be the number of PCA dimensions and the type of feature (which you will explore in the next part) with a fixed k. Hence, there is no need to try to make intuitive sense of your Dunn Index across different k.
Include: Discuss the results of your experiments, and include numerical values, where appropriate.

7. Now we will give you more sophisticated (i.e. more realistic) features to use for clustering the artworks. Other than the raw pixel values that you have been using so far, we provide you with GIST and CNN features, which you can load in using the `loadCSV` function by passing in the relevant CSV file (`gist_features.csv, deep_features.csv`). You can read more about these features in the appendix. You will look at the clustering results obtained with different features types and a few values of k. For simplicity, set $m = 200$ for all your experiments. Thus, you will vary only feature type (raw pixel features, GIST features, CNN features) and $k = [2, 5, 10]$. Consider how cluster quality, Dunn index, and cluster purity vary across feature type and k. Is there a feature type that works the best across all values of k?
Include: Discuss the results of your experiments, and include numerical values, where appropriate.

Semi-supervised classification

Now we consider a more realistic scenario, where we have very sparsely-labeled data - i.e., only a few artist labels, and the rest of the labels unknown. A classifier would have a very hard time learning a mapping from features to artist labels in this setting (why?). However, clustering can help us generalize from a few labelled examples.

8. Please use the K-means clustering algorithm with the CNN features and $m = 200$, and use the `init_medoids_plus` function for initialization. You will vary 2 parameters: $k = [10, 30, 50, 70]$ and the amount of labeled data. The provided function `classifyUnlabeledData` allows you to vary how much labeled data you have by setting the relevant parameter in the code. Note that to minimize variance in results, we have pre-generated partially labeled datasets, and thus you should specifically try the following specific parameter values: $5\%, 15\%, 25\%, 50\%, 75\%, 100\%$. The resulting clusters generated from your clustering algorithm will result in labeled artworks being distributed across clusters, and not all clusters may contain labeled data.

   The way this function produces a classification of the data is similar to the majority voting from before, where each cluster produced by the clustering algorithm is assigned an artist label based on the most frequently occurring label for artworks within that cluster. For example, if after clustering we see that 70% of the labeled data are 'Monet' and the other 30% are 'Rembrandt', then we would simply denote that cluster to be 'Monet'. The difference now is that very few of the artworks are labeled, so we compute the majority vote only over the labels we have. We also have to think harder about what to do in cases when there is no labeled data in a cluster, or when we have equal numbers of different labels. Look at the code inside the `classifyUnlabeledData` function: what classification decisions did we make in these cases?

This function outputs a classification accuracy which is the proportion of artworks that are correctly classified using this majority voting approach. Report the classification accuracy of the unlabeled points as it varies with k and the amount of labeled data. You might get slightly different results each time you run, so please average your accuracies over at least 3 iterations. What happens as you increase k and decrease the amount of labeled data at the same time? Should choice of k depend on the amount of labeled data?

Include: Discuss the results of your experiments, and include numerical values, where appropriate.

REMEMER Submit a .zip file containing your source code and your report in PDF format to Stellar.

Appendix: some background and details

Eigendecomposition using SVD

In the PCA you have seen in recitation, we computed the eigenvectors of the covariance matrix $X^T X$. Note, however, that applying SVD (singular value decomposition) to X, we get $X = U\Sigma V^T$; where U and V are unitary matrices, and Σ is a diagonal matrix with non-negative entries (called singular values). Thus, $X^T X = (U\Sigma V^T)^T(U\Sigma V^T) = (V\Sigma U^T)(U\Sigma V^T) = (V\Sigma^2 V^T)$ where the singular values of X are now the square roots of the eigenvalues of $X^T X$. Also the columns of V, which are known as right-singular vectors of X, are precisely the eigenvectors of $X^T X$. So, we can use SVD to compute the eigenvalues and eigenvectors of $X^T X$ without ever explicitly computing the covariance matrix. In practice, this is the method used for PCA.

Image features

GIST: this is a popular image descriptor, often used for scene classification[3]. To compute this image descriptor, an image is convolved with Gabor filters (linear filters used for edge detection in images) at different scales and orientations. The filter responses are then averaged over grid cells in the image and concatenated into a single feature vector. The GIST descriptors that we have provided are computed in the following way: 32 Gabor filters are convolved with the image to produce 32 filter response maps. Each map is then divided into a 4×4 grid, and the values are averaged within a grid cell. The resulting concatenated feature vector is thus $32 \times 16 = 512$-dimensional. GIST summarizes the gradient distributions in an image: it gives a good idea of where there is lots of texture (edges) in an image, and where there is more open space (lack of texture). Thus, it is used to describe the 'gist' of a scene.

CNN: neural networks have been around since the early days of artificial intelligence, modeled after the human brain: computational 'neurons' wired together hierarchically, capable of adapting to (learning from) new inputs, with each hierarchical layer learning progressively more complex representations, and the final layer solving complex recognition tasks. Convolutional neural networks (CNN) have recently proved very successful in computer vision for solving a variety of problems[4]. The features we provide you with have been extracted from one of the top layers of a CNN. We have given you a 4096-dimensional feature vector per image—you can think of each coordinate in the feature vector as encoding how likely some object category (e.g., human, dog, sky, etc.) is present in an image. The CNN features seem to be good for capturing object-level semantic content of an image.

Many other feature representations are possible. All of them are variations on the same theme: abstract away from raw pixels, and form more general representations of image content. This can be done by histogramming and averaging measurements in different image regions. These measurements are often pixel intensities (grayscale or color) or gradients (edges). Many different ways of computing gradient distributions are possible - with variations in how the gradients are measured, the histogram granularity, how the averaging is performed, what type of normalization is applied, etc. (some other examples of gradient-level features include **HOG** features, **SIFT** features, etc.). A histogram over an image region just specifies how much of each measurement there is in a region (e.g. how many horizontal edges, vertical edges, etc.) without specifying the exact pixel location. This allows for a more general representation that will still match images that have similar distributions of features but are not identical at the pixel level.

[3] Read more about GIST: `<http://people.csail.mit.edu/torralba/code/spatialenvelope/>`

[4] An accessible summary of some of the applications of CNNs to computer vision is provided here: `<http://arxiv.org/pdf/1403.6382.pdf>`